Amendments to the Specification:

In the specification, please replace page 2, paragraph 1, line 2, with the following:

Application, drivers, etc. are generally written in a high-level lanugage such as C. Such languages have typically been implemented primarily by compilation to native code. In such cases, drivers are written separately from the application and other programs that operate on a system. The application and drivers are then typically linked together either during an installation process or Dynamic Link Library (DLL) ~~(e.g., DLL)~~ when the application is executed. The advantage of such a system is that the compiler can be designed to optimize the code for a particular class of processor (e.g X86). However, the compiler may not optimize the code for a particular microprocessor, e.g., PENTIUM IV versus PENTIUM III. Moreover the compiler does not optimize the code for other system parameters including driver versions and other hardware components or take into account the particular system constraints of the target system. Instead, the application or runtime level system must employ computationally expensive logic to determine such parameters and processor constraints so that the program can be compiled to execute on an entire class of computer systems.

In the specification, please replace page 2, paragraph 2, line 10, with the following:

Another common programming paradigm is to compile code at runtime. A Just-In-Time (JIT) compiler is an example of such as system. Other systems that compile at runtime include continuous compilation systems that immediately begin execution in an interpretive state but compile the code over time and continously optimize the compilation. With just-in-time compilers, as classes are loaded into the virtual machine, the method pointers in the virtual method table are replaced with pointers to the JIT compiler. Then, the first time each method is called, the JIT compiler is invoked to compile the method. The pointer in the virtual method table is then patched to point to the native-code version of the method so that future calls to the method will jump to the native-code. These JIT compiler systems have the advantage of trasmitting code to a target machine in an intermediate langauge (IL) such as

JAVA bytecodes, Common Language Runtime (CLR) [[CLRT]] instructions, and so on. The compiler is designed to convert the IL into instructions executable by the native processor. As a result, the same IL instructions can be sent to computers having different native processors and execute nonetheless on the target processor.

In the specification, please replace page 4, paragraph 2, line 8, with the following:

The invention described herein recognizes that managed code, including applications, runtime, and driver, should have a priori knowledge of the client's exact hardware configuration, just as the JIT compiler has a priori knowledge of the microprocessor type on the client. For example, at JIT time, the system knows the effective version of the graphics driver (DIRECT~~Direct~~X 6.0, DIRECT~~Direct~~X 7.0, and so on), so if the application and driver are managed, the JIT compiler can emit an executable tuned for a particular driver version.

In the specification, please replace page 5, paragraph 2, line 3, with the following:

Figure 1 provides a schematic diagram of an exemplary networked or distributed computing environment. The distributed computing environment comprises computing objects 10a, 10b, etc. and computing objects or devices 110a, 110b, 110c, etc. These objects may comprise programs, methods, data stores, programmable logic, etc. The objects may comprise portions of the same or different devices such as Personal Digital Assistant (PDA) PDAs, televisions, Motion Picture Experts Group (MPEG) MPEG Audio Layer 3 (MP3) ~~MP3~~ players, televisions, personal computers, etc. Each object can communicate with another object by way of the communications network 14. This network may itself comprise other computing objects and computing devices that provide services to the system of Figure 1. In accordance with an aspect of the invention, each object 10 or 110 may contain data for which it would be desirable to perform image cut-out or boundary definition. It may also be desirable to compare an image cut-out from one object 10 or 110 with an image cut-out of another object 10 or 110.

In the specification, please replace page 6, paragraph 1, line 3, with the following:

It can also be appreciated that an object, such as 110c, may be hosted on another computing device 10 or 110. Thus, although the physical environment depicted may show the connected devices as computers, such illustration is merely exemplary and the physical

environment may alternatively be depicted or described comprising various digital devices such as PDAs, televisions, MP3 players, etc., software objects such as interfaces, Component Object Model (COM) [[COM]] objects and the like.

In the specification, please replace page 7, paragraph 3, line 14, with the following:

Thus, Figure 1 illustrates an exemplary networked or distributed environment, with a server in communication with client computers via a network/bus, in which the present invention may be employed. In more detail, a number of servers 10a, 10b, etc., are interconnected via a communications network/bus 14, which may be a Local Area Network (LAN) [[LAN]], Wide Area Network (WAN)[[WAN]], intranet, the Internet, etc., with a number of client or remote computing devices 110a, 110b, 110c, 110d, 110e, etc., such as a portable computer, handheld computer, thin client, networked appliance, or other device, such as a VCR, TV, oven, light, heater and the like in accordance with the present invention. It is thus contemplated that the present invention may apply to any computing device in connection with which it is desirable to communicate to another computing device with respect to image cut-out or boundary definition services.

In the specification, please replace page 13, paragraph 1, line 8, with the following:

For example, MICROSOFT®'s .Net platform includes servers, building-block services, such as Web-based data storage and downloadable device software. Generally speaking, the .Net platform provides (1) the ability to make the entire range of computing devices work together and to have user information automatically updated and synchronized on all of them, (2) increased interactive capability for Web sites, enabled by greater use of Extensible Markup Language (XML) [[XML]] rather than HTML, (3) online services that feature customized access and delivery of products and services to the user from a central starting point for the management of various applications, such as e-mail, for example, or software, such as Office .Net, (4) centralized data storage, which will increase efficiency and ease of access to information, as well as synchronization of information among users and devices, (5) the ability to integrate various communications media, such as e-mail, faxes, and telephones, (6) for developers, the ability to create reusable modules, thereby increasing

productivity and reducing the number of programming errors and (7) many other cross-platform integration features as well.

In the specification, please replace page 14, paragraph 2, line 10, with the following:

Figure 4 further illustrates the layers of an example application, runtime, and driver in a system. The Application 135, Runtime 302, and part of the Driver 303 operate in user mode to write drawing commands into hardware-specific command buffers in Direct Memory Access (DMA)[[DMA]] memory. In a contemporary PC system, these writes would typically be non-temporal writes into AGP memory; and as depicted in this implementation example, Application 135 resides in an EXE and Runtime 302 and User Mode Driver 303 reside in DLLs that are dynamically linked into Application 135. These details of the user mode portion of the system can vary; specifically, the Application 135, Application 135 and Runtime 302, or Application 301, Runtime 302 and User Mode Driver 303 could potentially be managed code.

In the specification, please replace page 18, paragraph 1, lines 3-4 , with the following:

In the context of the invention, however, the most important advantage of JIT compilation is that while the managed code is being generated, the JIT compiler has a priori knowledge of the exact nature of the target computer (i.e., the client the JIT compiler is running on). If the client computer has a particular type of microprocessor, the JIT compiler can emit code that is native to that specific microprocessor. For example, the ~~Pentium~~ PENTIUM Pro microprocessor added conditional move instructions to the x86 instruction set, and the ~~Pentium~~ PENTIUM 3 microprocessor added pre-fetch and other cache management instructions that were not available on its predecessors. Supporting these microprocessor-specific instructions in traditionally-deployed software requires the developer to write source code that uses all the various features, then write detection software to figure out which code path to execute on the client that the code happens to be running on. The JIT step frees the developer from this task and even proffers the developer protection against future innovations. In other words, a computer that included a new instruction that would benefit the developer's application could include a JIT compiler that knew how to emit that

instruction; the application would benefit from the new instruction even if it did not exist when the application was developed.

In the specification, please replace page 19, paragraph 4, line 20, with the following:

Figure 7 provides an alternate embodiment of the managed code system. Here, the architecture would enable the compiled Application 135 to write hardware-specific commands directly into command buffers or First-in first-outs (FIFOs)[[FIFOs]]. Besides the performance implications, other potential benefits include reducing the engineering effort required for IHVs to deliver cross-platform drivers and better enabling validation tools to ensure the drivers' correctness. The Application 135, Runtime 302, and Driver 303 are all delivered to JIT 602 in IL form. JIT 602 converts them into a Managed Application 604.